Package ‘dagR’

February 14, 2012

Type Package

Title R functions for directed acyclic graphs

Version 1.1.1

Date 2010-06-14

Author Lutz P Breitling

Maintainer Lutz P Breitling <lutz.breitling@gmail.com>

Description Functions to draw, manipulate and evaluate directed acyclic graphs.

License GPL-2

LazyLoad yes

Repository CRAN

Date/Publication 2010-06-21 06:38:17

R topics documented:

- dagR-package .................................................. 2
- add.arc ................................................. 3
- add.node ............................................. 4
- addAngle ............................................. 5
- allCombs ............................................. 6
- angle ............................................... 7
- anglePoint .......................................... 8
- assoc.exists ....................................... 9
- brute.search ....................................... 9
- dag.adjust ......................................... 10
- dag.adjustment .................................... 11
- dag.ancestors ..................................... 13
- dag.draw ........................................... 14
- dag.init ............................................ 15
- dag.legend .......................................... 16
Description

The package dagR (pronounce "dagger") contains a couple of functions to draw, manipulate and evaluate directed acyclic graphs (DAG), with a focus on epidemiologic applications, namely the assessment of adjustment sets and potentially biasing paths. The functions for finding and evaluating paths essentially implement the graphical algorithms outlined in Greenland (1999).

When using this package for your work, please cite Breitling (2010).

Note: As spelled out in the license, this suite of functions comes without any warranty, and cautious use is strongly advised. Although testing was carried out as meticulously as possible, it must be expected that bugs or errors remain, in particular in the early versions of the package. Please report any problems, concerns, but also suggestions for improvements or extensions to the author.

Important additions in future versions could be e.g. drawing routines allowing full labels of nodes in the DAG and algorithms with intelligent/efficient search for minimal adjustment sets.
add.arc

Details

Package: dagR
Type: Package
Version: 1.1.1
Date: 2010-06-14
License: GPL-2
LazyLoad: yes

dag.init is used for setting up DAGs. See the code of the functions demo.dag0 to demo.dag6 for example code. To adjust and/or evaluate DAGs for biasing paths, use dag.adjust, dag.draw for drawing a DAG. dag.search uses brute.search to evaluate all possible adjustment sets, allowing the identification of minimal sufficient adjustment sets using msas. At present, summary_dagRdag can summarize a DAG object. This should later become an S3 method. Several helper functions currently are not hidden and should later be made internal.

Please see the NEWS file for version changes and known open issues.

Author(s)

Lutz P Breitling <lutz.breitling@gmail.com>

References


Examples

dag1<-demo.dag1();
dag.draw(dag1);
dag1a<-dag.adjust(dag1, 3);
dag.draw(dag1a);
dag1s<-dag.search(dag1);
summary_dagRdag(dag1);
summary_dagRdag(dag1a);
summary_dagRdag(dag1s);

add.arc

Add an arc to a DAG.

Description

Function to conveniently add an arc to an existing DAG.
add.node

Usage

add.arc(dag, arc, type = 0)

Arguments

dag    The DAG to which an arc should be added.
arc    A vector of length 2, indicating from which node (first element) to which node
       (second element) the arc is to go. Note: the node numbering follows the num-
       ber of the existing DAG (as shown in dag.draw with option numbering=T),
       not the numbering of dag.init.
type   0 (=default) for a directed arc, 1 for an undirected association.

Value

A DAG with the arc (and corresponding arc.type) added, and with the path-related variables
(paths, pathsN, path.status, searchType, searchRes) removed.

Author(s)

Lutz P Breitling <lutz.breitling@gmail.com>

See Also

rm.arc, add.node, rm.node

Examples

dag1<-demo.dag1();
dag1b<-add.arc(dag1, arc=c(2,4));

add.node(Add a node to an existing DAG.

Description

Conveniently adds a node to an existing DAG, inserting its coordinates and label before the outcome
node. Also updates the arcs correspondingly.

Usage

add.node(dag, name = "unknown", type = 1, x = NA, y = NA)
addNode

Arguments

dag The DAG to which the node is to be added.
name Label for the node (defaults to "unknown").
type Type of node (1=covariable, 2=unknown); defaults to 1.
x X coordinate for the node position.
y Y coordinate for the node position.

Details

If no x and y coordinates are provided, the function places the node in an arbitrary position, slightly different with each additional node, so that one can more easily replace the nodes afterwards using dag.move.

Value

A DAG with the new node added.

Author(s)

Lutz P Breitling <lutz.breitling@gmail.com>

See Also

rm.node, add.arc, rm.arc

Examples

dag1<-demo.dag1();
dag1a<-add.node(dag1);

addAngle

Sum up two radian angles.

Description

Adds two radian angles together and applies modulus 2*pi. This is internally called by smoothArc, though hardly needed.

Usage

addAngle(a, b)

Arguments

a Angle 1 in radian.
b Angle 2 in radian.
allCombs

Value
numeric value [0, 2*pi).

Author(s)
Lutz P Breitling <lutz.breitling@gmail.com>

See Also
smoothArc

Examples
addAngle(/zero.noslash.5*pi, pi);
addAngle(1.5*pi, pi);

Description
Creates a matrix with all combinations of 1 to all elements of the vector provided. Elements to occur in all combinations can be specified. This is internally called by brute.search.

Usage
allCombs(x, force = c(), trace = FALSE)

Arguments
x A vector of elements of which combinations are to be formed.
force A vector of elements that are supposed to occur in each combination.
trace A boolean indicating if some output should be printed (TRUE) or not (FALSE=default).

Value
A matrix with one combination per row. For the shorter combinations, the columns to the right are filled up with NA.

Author(s)
Lutz P Breitling <lutz.breitling@gmail.com>

See Also
brute.search
angle

Examples

```r
allCombs(1:4, force=3);
```

---

angle Calculate radian angle of line between two points.

Description

Calculates the radian angle of the line connecting two points. Internally called by smoothArc.

Usage

```r
angle(A, B)
```

Arguments

- **A**: Vector of length two indicating the coordinates of the first point.
- **B**: Vector of length two indicating the coordinates of the second point.

Value

A numeric value [0, 2*pi).

Author(s)

Lutz P Breitling <lutz.breitling@gmail.com>

See Also

smoothArc, addAngle

Examples

```r
angle(c(0,0), c( 0,-1)) /pi;
angle(c(0,0), c(-1,-1)) /pi;
angle(c(0,0), c(-1, 0)) /pi;
angle(c(0,0), c(-1, 1)) /pi;
angle(c(0,0), c( 0, 1)) /pi;
angle(c(0,0), c( 1, 1)) /pi;
angle(c(0,0), c( 1, 0)) /pi;
angle(c(0,0), c( 1,-1)) /pi;
```
anglePoint  

*Calculate coordinates at specific angle and distance.*

**Description**

A function calculating the coordinates of the point that is at a specific radian angle in a specific distance from a source point. Internally called by smoothArc.

**Usage**

anglePoint(A, angl, len)

**Arguments**

- **A**  
  Vector of length two with the coordinates of the source point.

- **angl**  
  Radian angle indicating into which direction the new point is to be calculated.

- **len**  
  The distance at which the new point is situated from the source point.

**Value**

A vector of length two with the coordinates of the new point.

**Note**

Another pretty superfluous helper function...

**Author(s)**

Lutz P Breitling <lutz.breitling@gmail.com>

**See Also**

smoothArc

**Examples**

```plaintext
round(anglePoint(c(0,0), 0, 0.5), 2);
round(anglePoint(c(0,0), pi/2, 0.5), 2);
round(anglePoint(c(0,0), pi, 0.5), 2);
round(anglePoint(c(0,0), 1.5*pi, 0.5), 2);
```
assoc.exists

Check if association between two DAG nodes exists.

Description

Checks if an association between two DAG nodes already exists, i.e. does not need to be introduced when adjusting for a shared child etc. Internally called by \texttt{dag.adjustment}.

Usage

\texttt{assoc.exists(dag, a, b)}

Arguments

- \texttt{dag} The DAG to be dealt with.
- \texttt{a} First node.
- \texttt{b} Second node.

Value

A boolean indicating whether or not an association between first node and second node already exists.

Author(s)

Lutz P Breitling <lutz.breitling@gmail.com>

Examples

\begin{verbatim}
dag1a<-dag.adjust(demo.dag1(), A=3);
assoc.exists(dag1a, 2, 4);
assoc.exists(dag1a, 2, 5);
\end{verbatim}

brute.search

Evaluate all possible adjustment sets of a DAG.

Description

Evaluates all adjustment sets of a DAG, optionally including adjustment sets including "unknown" nodes. If the DAG has a non-empty adjustment set, only adjustment sets including these adjustment variables are evaluated.

Usage

\texttt{brute.search(dag, allow.unknown = FALSE, trace = TRUE, stop = 0)}
Arguments

- **dag**: The DAG to be evaluated.
- **allow.unknown**: Boolean indicating "unknown" nodes should be featured in the adjustment sets to be evaluated (TRUE) or not (FALSE=default).
- **trace**: Boolean indicating if some output should be produced (TRUE=default).
- **stop**: If =0, all eligible adjustment sets are evaluated. If =1, evaluations are stopped after the first sufficient adjustment set has been evaluated. Defaults to 0.

Value

A dataframe with the first columns (X1..Xn) indicating the variables in the respective adjustment set evaluated. The column `totalPaths` indicates the number of paths found when adjusting for the respective set, and `openPaths` indicates the number of biasing paths.

Note

The output produced by `brute.search` allows to manually identify sufficient and minimal sufficient adjustment sets, which in the future should preferably be done by a helper summary function. The evaluation of a complicated DAG like demo.dag2 can take quite some time, and future functions should either employ more intelligent algorithms to search specifically for sufficient sets, or they should allow e.g. the evaluation of adjustment sets of specific sizes.

Author(s)

Lutz P Breitling <lutz.breitling@gmail.com>

Examples

```r
bs <- brute.search(demo.dag1(), trace=FALSE);
bs;
```

---

### dag.adjust

Adjust an existing DAG for covariables.

Description

The function looks for associations introduced by adjusting for the covariables specified, then looks for biasing paths, and finally evaluates these paths.

Usage

```r
dag.adjust(dag, A = c())
```
**Arguments**

- **dag**: The DAG to be adjusted (or evaluated).
- **A**: Vector indicating the adjustment set. *The numbering is according to the nodes vector of the DAG, which is shown e.g. in the legend of a DAG drawn by dag.draw. This numbering is different (+1) from the one used in dag.init, because the nodes vector also contains the exposure at position 1 (in contrast to the covariables vector used in dag.init)!*

**Details**

If the adjustment set is empty, the function only looks for biasing paths and evaluates these.

**Value**

A DAG with the adjustment set A, and possibly with additional associations introduced by adjustment, biasing paths found, and the status of these.

If adjustment set is not empty, searchType and searchRes are set to NULL.

**Note**

CAVE: Do not apply this to an already adjusted DAG, since this might not be handled appropriately (see documentation of dag.adjustment called by dag.adjust).

**Author(s)**

Lutz P Breitling <lutz breitling@gmail.com>

**See Also**

- `dag.adjustment`
- `find.paths`
- `eval.paths`

**Examples**

```r
demo.dag1();
dag.adjust(demo.dag1());
dag.adjust(demo.dag1(), A=3);
```

---

**Description**

Identifies the associations introduced by adjustment for the variables specified, and returns the DAG with these associations added. Note that this is called internally by dag.adjust, which makes sure that biasing paths are looked for and evaluated afterwards. Thus, dag.adjustment should 1.) not be called directly, and 2.) not be called on an already adjusted DAG!
dag.adjustment

Usage

dag.adjustment(dag, A=NULL)

Arguments

dag
The DAG to be adjusted.
A
The adjustment set to be applied.

Details

The adjustment set A specified when calling `dag.adjustment` overrules the adjustment variables that are present in the DAG. To keep these in the adjustment set, one has to add them to A.

Value

A DAG with A as the adjustment set and the associations introduced by adjustment for A added to the DAG.

Note

You should not use `dag.adjustment` on an already adjusted DAG, since it cannot identify associations that had been introduced by the earlier adjustment. If the new adjustment set does not include the adjustment variables present in the first set, the new DAG might feature associations that actually only would be introduced when adjusting for the variables featured in the first but not second adjustment set.

Author(s)

Lutz P Breitling <lutz.breitling@gmail.com>

See Also

dag.adjust, find.paths, eval.paths

Examples

dag1<-demo.dag1();
dag.draw(dag.adjust(dag1, 3));
dag.draw(dag.adjust(dag1, 2));
# problem when adjusting an adjusted DAG:
dag.draw(dag.adjust(dag.adjust(dag1, 3), c(2)));
**Description**

This identifies those nodes in a DAG that are ancestors of the nodes specified, i.e. acc. to the model depicted by the DAG they causally precede those nodes. Internally called by `dag.adjustment` in the context of finding associations introduced by adjustment.

**Usage**

`dag.ancestors(dag, A)`

**Arguments**

- `dag` The DAG to be evaluated.
- `A` A vector of nodes for which ancestors are to be identified.

**Value**

A vector indicating which nodes are ancestors of those in `A`. Note that `A` actually is included at the beginning of the vector.

**Author(s)**

Lutz P Breitling <lutz.breitling@gmail.com>

**See Also**

`dag.adjust, dag.adjustment`

**Examples**

```r
dag6<-demo.dag6();
dag.draw(dag6);
dag.ancestors(dag6,4);
dag.ancestors(dag6,3);
dag.ancestors(dag6,2);
dag.ancestors(dag6,1);
dag.ancestors(dag6,7);
```
Description

Draws a DAG conforming to the dagR format. The nodes are represented by 'C' (covariables; numbered with subscripts) and 'U' (unknown/unmeasured covariables; numbered with subscripts), 'X' and 'Y' (exposure and outcome, respectively). A legend presents the names of the nodes. The X->Y arc is marked with a questionmark as the relationship of interest. Adjusted variables are under- and over-lined. Undirected associations are drawn with dashed lines. If paths have been identified (and evaluated), these (and their status) are written next to the legend.

Usage

dag.draw(dag, legend = TRUE, paths = TRUE, numbering = FALSE, p = FALSE, ...)

Arguments

dag The DAG to be drawn.
legend Boolean indicating whether a node legend should be included.
paths Boolean indicating whether paths (and their status) should be written.
numbering Boolean indicating whether the arcs should be numbered in the DAG.
p Boolean indicating whether the curving points of undirected associations should be drawn.
... Currently not used.

Value

Returns the DAG (for whatever reason...).

Author(s)

Lutz P Breitling <lutz.breitling@gmail.com>

References


See Also

dag.letter, garrows, smoothArc, dag.legend, write.paths
**Examples**

```r
dag.draw(demo.dag1());
dag.draw(dag.adjust(demo.dag1(), 3), numbering=TRUE, p=TRUE);
```

---

**Set up a new DAG.**

**Description**

Allows setting up a new DAG. See the `demo.dag0` to `demo.dag6` functions for some example specifications.

**Usage**

```r
dag.init(outcome = NULL, exposure = NULL, covs = c(), arcs = c(),
  assocs = c(), xgap = 0.04, ygap = 0.05, len = 0.1, y.name = NULL,
  x.name = NULL, cov.names = c(), ...)
```

**Arguments**

- `outcome`: Currently not used!
- `exposure`: Currently not used!
- `covs`: Vector including an integer for each covariable to be in the DAG (1 for a "standard" covariable, 2 for an unknown/unmeasured one).
- `arcs`: Vector of duplets of integers, in which nodes from which an arc or undirected association is to emanate are followed by those to which it is to point. To refer to the exposure, use 0, to refer to the outcome, use -1, to refer to covariables, use and element of 1:length(covs).
- `assocs`: A vector of same length as covs, with 0 indicating directed arcs, 1 indicating undirected associations.
- `xgap`: How much x space is to be left between arc ends and nodes when drawing?
- `ygap`: How much y space is to be left between arc ends and nodes when drawing?
- `len`: Length of arrow whiskers when drawing.
- `y.name`: Label of outcome.
- `x.name`: Label of exposure.
- `cov.names`: Vector of covariable labels.
- `...`: Currently not used.

**Value**

A DAG (objects of class `dagRdag`). Check out some of the demonstration DAGs for details. The DAG is actually a list object, with elements `cov.types` (the `covs` vector, with 0 put in front, and -1 at the end); `x` and `y` (coordinates for drawing the nodes, initially set up more or less in a half-circle above the x->y arc); `arc` (the arcs, transformed into a matrix); `arc.type` (the `assocs` vector); `curve.x` and `curve.y` (if associations are featured, these provide the coordinates through which to curve); `xgap`, `ygap`, `len` (the respective drawing parameters); `version` (dagR version).
Note

CAVE: The numbering of the covariables and arc coordinates is different here than in the functions later used on the DAG (e.g. add.arc, dag.adjust)! The functions generally work according to the indexing of the R objects that they handle. Whereas for dag.init the n covariable nodes are numbered 1:n, the node vector of the resulting DAG will also contain the exposure node at the beginning and the outcome node at the end, i.e. it will go from 1:(n+2) with the covariables at 2:n+1. summary_dagRdag will show the latter numbering. Example: when adjusting for the first covariable, dag.adjust must be handed the adjustment set A=2, as the first covariable will occupy the second node (the first node is occupied by the exposure).

Author(s)

Lutz P Breitling <lutz.breitling@gmail.com>

References


See Also

dag.draw

demo.dag1;
demo.dag1();
demo.dag2;
demo.dag2();

dag.legend

Description

Lists the DAG symbols along with their names/labels below a DAG drawn.

Usage

dag.legend(dag, lx = -0.15, ly = -0.075)

Arguments

dag The DAG for which the legend is needed.
lx X coordinate for repositioning legend.
ly Y coordinate for repositioning legend.
**Author(s)**

Lutz P Breitling <lutz.breitling@gmail.com>

**See Also**

`dag.draw`, `write.paths`

**Examples**

```r
dag.draw(demo.dag1(), legend=FALSE);
dag.legend(demo.dag1(), lx=0.2, ly=0.01);
```

---

**Description**

Writes the node symbols, 'X' and 'Y' for exposure and outcome, 'C' and 'U' (with consecutive subscripts) for known and unknown covariables. Unknownness is identified by either node name 'unknown' or covariable type '2' in the DAG object.

**Usage**

```r
dag.letter(dag, letter, x, y)
```

**Arguments**

- `dag`: The DAG for which a node is to be written.
- `letter`: The node that is of interest.
- `x`: X position.
- `y`: Y position.

**Author(s)**

Lutz P Breitling <lutz.breitling@gmail.com>

**See Also**

`dag.draw`, `dag.legend`, `write.paths`

**Examples**

```r
dag2a<-dag.adjust(demo.dag2(), 3);
dag.draw(dag2a, paths=FALSE);
dag.letter(dag2a, 1, 0.2, 1);
dag.letter(dag2a, 2, 0.25, 1);
dag.letter(dag2a, 3, 0.3, 1);
dag.letter(dag2a, 4, 0.35, 1);
dag.letter(dag2a, 13, 0.4, 1);
```
**dag.move**

Interactively move a node or curving point in a DAG.

**Description**

This allows to reposition a node or association curving point of a DAG graphically. First, select a node or curving point by left-clicking close to it. Then reposition it to any other position by left-clicking. Once you are happy with the new position, right-click to exit.

**Usage**

```
dag.move(dag)
```

**Arguments**

- `dag` The DAG to be modified.

**Value**

The same DAG, but with the feature repositioned.

**Author(s)**

Lutz P Breitling <lutz.breitling@gmail.com>

**Examples**

```r
## Not run: dag1m<-dag.move(dag.adjust(demo.dag1(), 3));
```

---

**dag.search**

Evaluate possible adjustment sets of a DAG.

**Description**

Currently, this simply is a wrapper for `brute.search`, which returns the input DAG with the results of `brute.search` and a string describing the search setup.

**Usage**

```
dag.search(dag, type = "brute", allow.unknown = FALSE, trace = FALSE, stop = 0)
```
**Arguments**

- **dag**
  - DAG to be evaluated.
- **type**
  - Type of search to be performed. Currently, only =brute is possible.
- **allow.unknown**
  - See `brute.search`.
- **trace**
  - See `brute.search`.
- **stop**
  - See `brute.search`.

**Value**

The DAG with components `searchType` and `searchRes` added.

**Author(s)**

Lutz P Breitling <lutz.breitling@gmail.com>

**See Also**

- `brute.search`

**Examples**

```r

d3<-demo.dag3();
d3s<-dag.search(d3);
d3s;
```

---

**demo.dag0**

*Set up demo DAG #0.*

**Description**

Initializes a simple DAG used during the dagR development phase.

**Usage**

```r
demo.dag0()
```

**Value**

Returns a DAG.

**Author(s)**

Lutz P Breitling <lutz.breitling@gmail.com>

**See Also**

demo.dag1, demo.dag2, demo.dag3, demo.dag4, demo.dag5, demo.dag6
**Examples**

demo.dag0();

demo.dag1

*Set up demo DAG #1.*

**Description**

Initializes a classical "M DAG" useful for demonstrating harmful adjustment. The DAG is motivated by figure 3 in Fleischer (2008) and also featured in Breitling (2010).

**Usage**

demo.dag1()

**Value**

Returns a DAG.

**Author(s)**

Lutz P Breitling <lutz.breitling@gmail.com>

**References**


**See Also**

demo.dag0, demo.dag2, demo.dag3, demo.dag4, demo.dag5, demo.dag6

**Examples**

demo.dag1();
**demo.dag2**

Set up demo DAG #2.

---

**Description**

Initializes a more complex DAG, motivated by Shrier (2008). This DAG was used to examine the performance of `brute.search` and has been featured in Breitling (2010).

**Usage**

demo.dag2()

**Value**

Returns a DAG.

**Author(s)**

Lutz P Breitling <lutz.breitling@gmail.com>

**References**


**See Also**

demo.dag0, demo.dag1, demo.dag3, demo.dag4, demo.dag5, demo.dag6

**Examples**

demo.dag2();

---

**demo.dag3**

Set up demo DAG #3.

---

**Description**

Initializes a DAG motivated by the manual for the software DAG v0.11 (Kn"uppel 2009). This DAG has been featured in Breitling (2010).

**Usage**

demo.dag3()
demo.dag4

Value

Returns a DAG.

Author(s)

Lutz P Breitling <lutz.breitling@gmail.com>

References


See Also

demo.dag0, demo.dag1, demo.dag2, demo.dag4, demo.dag5, demo.dag6

Examples

demo.dag3();

---

demo.dag4

Set up demo DAG #4.

Description

Initializes a miscellaneous DAG. What happens if you adjust for the exposure’s child?

Usage

demo.dag4()

Value

Returns a DAG.

Author(s)

Lutz P Breitling <lutz.breitling@gmail.com>

See Also

demo.dag0, demo.dag1, demo.dag2, demo.dag3, demo.dag5, demo.dag6

Examples

demo.dag4();
x4<-dag.adjust(demo.dag4(), A=3);
**demo.dag5**

---

**Set up demo DAG #5.**

**Description**

Initializes a miscellaneous DAG. What happens if you adjust for the outcome’s child?

**Usage**

demo.dag5()

**Value**

Returns a DAG.

**Author(s)**

Lutz P Breitling <lutz.breitling@gmail.com>

**See Also**

demo.dag0, demo.dag1, demo.dag2, demo.dag3, demo.dag4, demo.dag6

**Examples**

demo.dag5();

x5<-dag.adjust(demo.dag5(), A=3);

---

**demo.dag6**

---

**Set up demo DAG #6.**

**Description**

Initializes a miscellaneous DAG. What happens if you adjust for the collider?

**Usage**

demo.dag6()

**Value**

Returns a DAG.

**Author(s)**

Lutz P Breitling <lutz.breitling@gmail.com>
distPoints

See Also
demo.dag0, demo.dag1, demo.dag2, demo.dag3, demo.dag4, demo.dag5

Examples
demo.dag6();
x6<-dag.adjust(demo.dag6(), A=3);

distPoints

Calculate distance between two points.

Description
Another rather superfluous helper function, internally used by smoothArc. Calculates the distance between two points.

Usage
distPoints(A, B)

Arguments
A Vector of length two, indicating x and y of first point.
B Vector of length two, indicating x and y of second point.

Value
Distance between the two points.

Author(s)
Lutz P Breitling <lutz.breitling@gmail.com>

Examples
dp<-distPoints(c(0,0), c(1,1));
dp;
dp^2;
eval.paths

Evaluate potentially biasing paths in a DAG.

Description
This essentially implements the graphical algorithm described in Greenland (1999) to identify open "backdoor" (or not strictly backdoor, but potentially biasing) paths in a DAG. Paths are identified as being 'open', 'blocked by collider', or 'blocked by adjustment'. If both latter conditions apply, 'blocked by collider' is returned.

Usage
eval.paths(dag)

Arguments
dag A DAG to which find.paths has already been applied (e.g. within dag.adjust).

Details
This function identifies a collider-blocked path as 'blocked by collider' even if it has been unblocked by adjusting for the collider. One could argue that this should not be the case. However, the biasing seems to be sufficiently represented in the DAG by the introduction of the association "jumping" the collider and potentially opening biasing paths.

Value
A DAG with component path.status added.

Author(s)
Lutz P Breitling <lutz.breitling@gmail.com>

References

See Also
dag.adjust, find.paths
Examples

```r
dag1 <- demo.dag1();
dag1a <- dag.adjustment(dag1, k=3); # normally called via dag.adjust();
dag1f <- find.paths(dag1a);
dag1e <- eval.paths(dag1f);
names(dag1);
names(dag1a);
names(dag1f);
names(dag1e);
```

Description

This identifies paths linking exposure and outcome in a DAG. Forward paths (including a directed arc emanating from the exposure) are not identified.

Usage

```r
find.paths(dag)
```

Arguments

- **dag**: A DAG for which paths should be found.

Value

A DAG with components `pathsN` (number of paths identified) and `paths` (matrix with each row describing one path by indicating the arcs forming the path; ends with `NA` as some other function recognize the end of the path that way) added.

Author(s)

Lutz P Breitling <lutz.breitling@gmail.com>

References


See Also

dag.adjust, eval.paths
Examples

demo.dag1();
find.paths(demo.dag1());

dag.draw(demo.dag1());

garrows(x/zero.noslash, y/zero.noslash, x1, y1, xgap, ygap, len = /zero.noslash.1)

Arguments

x0 X coordinate of origin.
y0 Y coordinate of origin.
x1 X coordinate of target node.
y1 Y coordinate of target node.
xgap Space between node and arc ends on x axis.
ygap Space between node and arc ends on y axis.
len Length of arrow whiskers (default=0.1).

Author(s)

Lutz P Breitling <lutz.breitling@gmail.com>

See Also
dag.draw, smoothArc

Examples

dag.draw(demo.dag1());
garrows(0,0, 0.5, 0.5, xgap=0.1, ygap=0.1, len=0.1);
garrows(0,0, 0.5, 0.6, xgap=0.1, ygap=0.1, len=0.3);
garrows(0,0, 0.5, 0.6, xgap=0.01, ygap=0.1, len=0.3);
inAngle  \hspace{1cm} \textit{Calculate angle between two arcs.}

**Description**

Another rather superfluous helper function, calculating the radian angle between two radian angles. Internally called by \textit{smoothArc}.

**Usage**

\texttt{inAngle(a, b)}

**Arguments**

\begin{itemize}
\item \texttt{a} \hspace{1cm} \text{Radian angle 1.}
\item \texttt{b} \hspace{1cm} \text{Radian angle 2.}
\end{itemize}

**Value**

Numeric in range from -\pi to \pi.

**Author(s)**

Lutz P Breitling <lutz.breitling@gmail.com>

**See Also**

\textit{smoothArc}

**Examples**

\begin{verbatim}
\texttt{inAngle(pi, pi)};
\texttt{inAngle(pi, -0.5*\pi)/\pi};
\texttt{inAngle(pi, 0.5*\pi)/\pi};
\texttt{inAngle(pi, 2.5*\pi)/\pi};
\end{verbatim}

\begin{center}
\textbf{is.acyclic}  \hspace{1cm} \textit{Check if a DAG actually is acyclic.}
\end{center}

**Description**

This function checks for each node in a DAG whether backtracing arcs leading to it results in an "infinite recursion" error indicating that there actually is a cyclic part in the DAG (which then obviously seems not to be a DAG).
Usage

is.acyclic(dag, maxSecs=NA)

Arguments

dag The DAG to be check.
maxSecs maximum time before function aborts;

Value

A list with two elements. acyclic is a boolean indicating whether the DAG is acyclic (=TRUE) or contains a cyclic component (=FALSE). nodewise is a vector containing 1 boolean per node in the DAG, TRUE indicating that backtracing from this node does not lead to a cyclic component, FALSE indicating that backtracing from this node leads to a cyclic component.

Author(s)

Lutz P Breitling <lutz.breitling@gmail.com>

Examples

dag6<-demo.dag6();
is.acyclic(dag6);
dag6c<-add.arc(dag6, c(3,5));
is.acyclic(dag6c);

is.in

Check if a specific numeric value occurs in a vector.

Description

Another trivial helper function, called internally by eval.paths. It checks whether the specified (numeric) value is part of a specified vector of (numeric) values.

Usage

is.in(x, c = NULL)

Arguments

x A numeric value, for which the presence in a vector is to be checked.
c A vector of numeric values.

Value

Boolean; TRUE if value is present, FALSE if not.
is.unknown

Check if a DAG node presents an unknown variable.

Description

Another helper function, internally used by brute.search. It checks whether the node specified is of type=2 or is named 'unknown'.

Usage

is.unknown(x, dag)

Arguments

x The node of interest.
dag The DAG to be evaluated.

Value

TRUE if unknown (acc. to type or name), FALSE otherwise.

Author(s)

Lutz P Breitling <lutz.breitling@gmail.com>

See Also

brute.search

Examples

dag2<-demo.dag2();
is.unknown(2, dag2);
is.unknown(3, dag2);
dag2u<-dag2; dag2u$names[2]<-"unknown";
is.unknown(2, dag2u);
is.unknown(3, dag2u);
Identify minimal sufficient adjustment sets.

Description

Evaluates DAG adjustment sets identified by a `dag.search` (or `brute.search`) for minimal sufficiency by counting for each sufficient adjustment set A how many smaller sufficient ones that are contained in A exist.

Usage

```r
msas(adjSets)
```

Arguments

- `adjSets` The `searchRes` component of a DAG (or the output of `brute.search`, which is used by `dag.adjust` to produce `searchRes`).

Value

A vector containing a -1 for each insufficient adjustment set, and for sufficient ones the number of smaller sufficient ones contained in it.

Author(s)

Lutz P Breitling <lutz.breitling@gmail.com>

References


See Also

`viv`, `summary_dagRdag`

Examples

```r
d3<-demo.dag3();
d3s<-dag.search(d3);
msas(d3$s$searchRes);
bs<-brute.search(d3);
msas(bs);
```
**Description**

Function to conveniently remove an arc from an existing DAG.

**Usage**

```r
rma.arc(dag, arc)
```

**Arguments**

- `dag` The DAG from which to remove the arc.
- `arc` A single integer, indicating which arc is to be removed (referring to the respective row of the `dag$arc` matrix).

**Value**

A DAG with the arc specified removed along with the corresponding attributes like arc types, curves, and path evaluation variables.

**Note**

The numbering of the arcs can be visualized by applying `dag.draw` with the option "numbering=TRUE".

**Author(s)**

Lutz P Breitling <lutz.breitling@gmail.com>

**See Also**

`add.arc`, `add.node`, `rm.node`

**Examples**

```r
dag1<-demo.dag1();
dag1$arc;
dag1rmarc<-rm.arc(dag1, 2);
```
rm.node

Remove a node from a DAG.

Description

Function to conveniently remove a node from an existing DAG.

Usage

rm.node(dag, node)

Arguments

dag
  The DAG from which to remove the node.

node
  A single integer, indicating which node is to be removed.

Value

A DAG with the node specified removed, along with the corresponding attributes and dependent variables, i.e. arcs involving this node are also removed, and the numbering of the nodes (and their occurrence in arcs) is corrected accordingly.

Note: Search components (searchType, searchRes) of the DAG currently are generally set to NULL, even if no path is removed. This is for simplicity, because the node numbers would need to be changed eg. in the searchRes variables etc.

Author(s)

Lutz P Breitling <lutz.breitling@gmail.com>

See Also

add.node, rm.arc, add.arc

Examples

dag1a<-dag.adjust(demo.dag1());
dag1armn<-rm.node(dag1a, 3);
smoothArc

Draw an undirected association in a DAG.

Description
This draws a dashed connection between two points, curving it so that it goes through a third point. This is internally used by dag.draw to draw associations.

Usage
smoothArc(A, B, C, res = 20, gap = 0.05, p = FALSE)

Arguments
A Vector of length 2, providing xy coordinates of first point.
B Vector of length 2, providing xy coordinates of second point.
C Vector of length 2, indicating xy coordinates through which the association should be curved.
res How smooth should the curve be drawn?
gap How far from point A and B should the line end?
p If TRUE, the point through which the curve goes is drawn (this is to allow better moving it with dag.move).

Note
In the version 1.0.1 distributed as online supplemental material with Breitling (2010), the function contains arbitrary default values used during development.

Author(s)
Lutz P Breitling <lutz.breitling@gmail.com>

References

See Also
dag.draw, dag.move

Examples
dag.draw(demo.dag1());
smoothArc(c(0.5,0.5), c(1,1), c(0.75,0.6), p=TRUE);
smoothArc(c(0.5,0.5), c(1,1), c(0.65,0.6), p=TRUE);
smoothArc(c(0.5,0.5), c(1,1), c(0.55,0.6), p=TRUE);
smoothArc(c(0.5,0.5), c(1,1), c(0.45,0.6), p=TRUE);
Summary 

Summarize a DAG.

Description

Generic function summary code for class dagRdag, which is used by package dagR from version 1.1.1 on. At present, it actually is NOT coded as a generic function!

Usage

summary_dagRdag(dag)

Arguments

dag An object of class dagRdag.

Details

Summarizes according to what functions have been applied to the DAG. It does not itself call dag.search and the like. Exception: is calls is.acyclic (with maxSecs=5).

Author(s)

Lutz P Breitling <lutz.breitling@gmail.com>

References


Examples

d3<-demo.dag3();
d3s<-dag.search(d3);
d3a<-dag.adjust(d3, 2);
d3as<-dag.search(d3a);
summary_dagRdag(d3);
summary_dagRdag(d3s);
summary_dagRdag(d3a);
summary_dagRdag(d3as);
viv  

Is a numeric vector in another vector?

Description

Checks if all numeric elements of a vector occur also in another vector. It is internally used by msas to check if some adjustment set is contained in another one.

Usage

viv(v1, v2)

Arguments

v1  
The vector whose occurrence in v2 is to be checked.

v2  
The vector in which v1 might occur.

Details

If a value occurs more than once in v1, it is counted as contained in v2 if it appears there once. An empty v1 (consisting only of NA) is considered to be contained in any v2.

Value

TRUE if v1 occurs in v2, FALSE otherwise.

Author(s)

Lutz P Breitling <lutz.breitling@gmail.com>

See Also

msas

Examples

v1<-c(NA,NA,NA)
v2<-c(1,NA,NA)
v3<-c(1,1,1)
viv(v1,v2)
viv(v2,v3)
viv(v2,v1)
write.paths

Write the paths into a DAG drawing.

Description
This function writes the paths into a DAG drawing, using the symbols ('C', 'U', 'X', 'Y') used in the drawing, indicating directed arcs by '<' and '>', undirected ones by '-'. Adjusted variables are under- and over-lined. If the paths have been evaluated using eval.paths, the status are also written.

Usage
write.paths(dag, px = 0.5, py = -0.06)

Arguments
dag The DAG that has been drawn.
px An x coordinate to change the position of the path writing.
py A y coordinate to change the position of the path writing.

Author(s)
Lutz P Breitling <lutz.breitling@gmail.com>

See Also
dag.draw, find.paths, eval.paths, dag.legend

Examples
dag1<-demo.dag1();
dag.draw(dag1);
dag1a<-dag.adjust(dag1);
write.paths(dag1a);
Index

*Topic package
   dagR-package, 2
   add.arc, 3, 5, 12, 13, 25, 26
   add.node, 4, 6, 32, 33
   addAngle, 5, 7
   allCombs, 6
   angle, 7
   anglePoint, 8
   assoc.exists, 9
   brute.search, 3, 6, 9, 19, 30
   dag.adjust, 3, 10, 12, 13, 25, 26
   dag.adjustment, 9, 11, 11, 13
   dag.ancestors, 13
   dag.draw, 3, 14, 16, 17, 27, 34, 37
   dag.init, 3, 15
   dag.legend, 14, 16, 17, 37
   dag.letter, 14, 17
   dag.move, 18, 34
   dag.search, 3, 18
   dagR (dagR-package), 2
   dagR-package, 2
   demo.dag0, 19
   demo.dag1, 20
   demo.dag2, 21
   demo.dag3, 21
   demo.dag4, 22
   demo.dag5, 23
   demo.dag6, 23
   distPoints, 24
   eval.paths, 11, 12, 25, 26, 30, 37
   find.paths, 11, 12, 25, 26, 37
   garrows, 14, 27
   inAngle, 28
   is.acyclic, 28
   is.in, 29
   is.unknown, 30
   msas, 3, 31, 36
   rm.arc, 4, 5, 32, 33
   rm.node, 4, 5, 32, 33
   smoothArc, 6–8, 14, 27, 28, 34
   summary_dagRdag, 31, 35
   viv, 31, 36
   write.paths, 14, 17, 37